CLAIMS

What is claimed is:

1. A system, comprising:

a first processor;

a second processor coupled to the first processor, the second processor having a core and comprising stack storage residing in the core;

memory coupled to, and shared by, the first and second processors; and

a synchronization unit coupled to the first and second processors, said synchronization unit synchronizes the execution of the first and second processors;

wherein the second processor executes stack-based instructions while the first processor executes one or more tasks wherein the first processor manages the memory via an operating system that executes only on the first processor and the first processor executes a virtual machine that controls the execution of a program on the second processor.

2. The system of claim 1 wherein the second processor comprises an internal data memory that holds a contiguous block of memory defined by an address stored in a register, and wherein local variables are stored in said data memory.

3. The system of claim 2 wherein the second processor executes methods and wherein when a new method is called by the second processor, the local variables associated with the called method use data memory space previously used by local variables associated with completed methods without generating a miss.

4. The system of claim 1 wherein the first processor executes a transaction targeting a pre-determined address and the synchronization unit detects said pre-determined address and asserts a wait signal to cause said first processor to enter a wait mode.

5. The system of claim 1 wherein the stack-based instructions comprise Java bytecodes and the first processor comprises a RISC processor so that the RISC processor executes one or more tasks while the second processor executes Java code.

6. The system of claim 1 further including a main stack residing outside the second processor's core and coupled to the stack storage in the second processor's core.

7. The system of claim 6 wherein the stack storage in the second processor's core provides an operand to execute a stack-based instruction in the second processor.

8. The system of claim 6 further including data flags that indicate coherence between the main stack outside the second processor's core and data within the stack storage in the second processor's core.

9. The system of claim 8 wherein coherency is established by examining the data flags and updating the main stack with values from the stack storage.

10. A method, comprising:

synchronizing the execution of first and second processors, the second processor having a core and comprising stack storage residing in the core;

executing stack-based instructions in the second processor while the first processor executes one or more tasks;

executing an operating system on the first processor and not on the second processor;

executing a virtual machine on the first processor that controls the execution of a program on the second processor; and

the first processor managing memory accessible to both the first and second processors via the operating system.

11. The method of claim 10 further including storing local variables in an internal data memory in the second processor, the internal data memory configured to store a contiguous block of memory defined by an address stored in a register.

12. The method of claim 11 further including executing methods on the second processor and wherein when a new method is called by the second processor, using, for the local variables associated with the called method, data memory space previously used by local variables associated with completed methods without generating a miss

13. The method of claim 11 wherein synchronizing comprises detecting that the first processor is executing a transaction targeting a pre-determined address and asserting a wait signal to cause said first processor to enter a wait mode.

14. The method of claim 11 further comprising providing a main stack residing outside the second processor's core and providing an operand from the stack storage in the second processor's core and executing a stack-based instruction in the second processor using the operand.

15. The method of claim 11 further comprising providing a main stack residing outside the second processor's core and asserting data flags that indicate coherence between the main stack outside the second processor's core and data within the stack storage in the second processor's core.

16. The method of claim 15 further determining coherency between the processors based on the data flags.

17. A system, comprising:

a first processor;

a second processor coupled to the first processor, the second processor having a core and comprising stack storage residing in the core and having an internal data memory that holds a contiguous block of memory defined by an address stored in a register, and wherein local variables are stored in said data memory; and

memory coupled to, and shared by, the first and second processors;

wherein the second processor executes stack-based instructions while the first processor executes one or more tasks wherein the first processor manages the memory via an operating system that executes only on the first processor and the first processor executes a virtual machine that controls the execution of a program on the second processor.

18. The system of claim 17 further comprising a synchronization unit coupled to the first and second processors, said synchronization unit synchronizes the execution of the first and second processors.

19. The system of claim 17 wherein the second processor executes methods and wherein when a new method is called by the second processor, the local variables associated with the called method use data memory space previously used by local variables associated with completed methods without generating a miss.